Capítulo 2: Camada de Aplicação

Baseado nos slides de Kurose e Ross

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

Capítulo 2: Camada de Aplicação

Metas do capítulo:

- aspectos conceituais e de implementação de protocolos de aplicação em redes
  - modelos de serviço da camada de transporte
  - paradigma cliente servidor
  - paradigma *peer-to-peer*
- aprender sobre protocolos através do estudo de protocolos populares da camada de aplicação:
  - HTTP
  - FTP
  - SMTP/ POP3/ IMAP
  - DNS
- Criar aplicações de rede
  - programação usando a API de *sockets*

Algumas aplicações de rede

- Correio eletrônico
- A Web
- Mensagens instantâneas
- Login em computador remoto como Telnet e SSH
- Compartilhamento de arquivos P2P
- Jogos multiusuários em rede
- *Streaming* de vídeos armazenados (YouTube, Hulu, Netflix)
- Telefonia por IP (Skype)
- Videoconferência em tempo real
- Busca
- ...
- ...

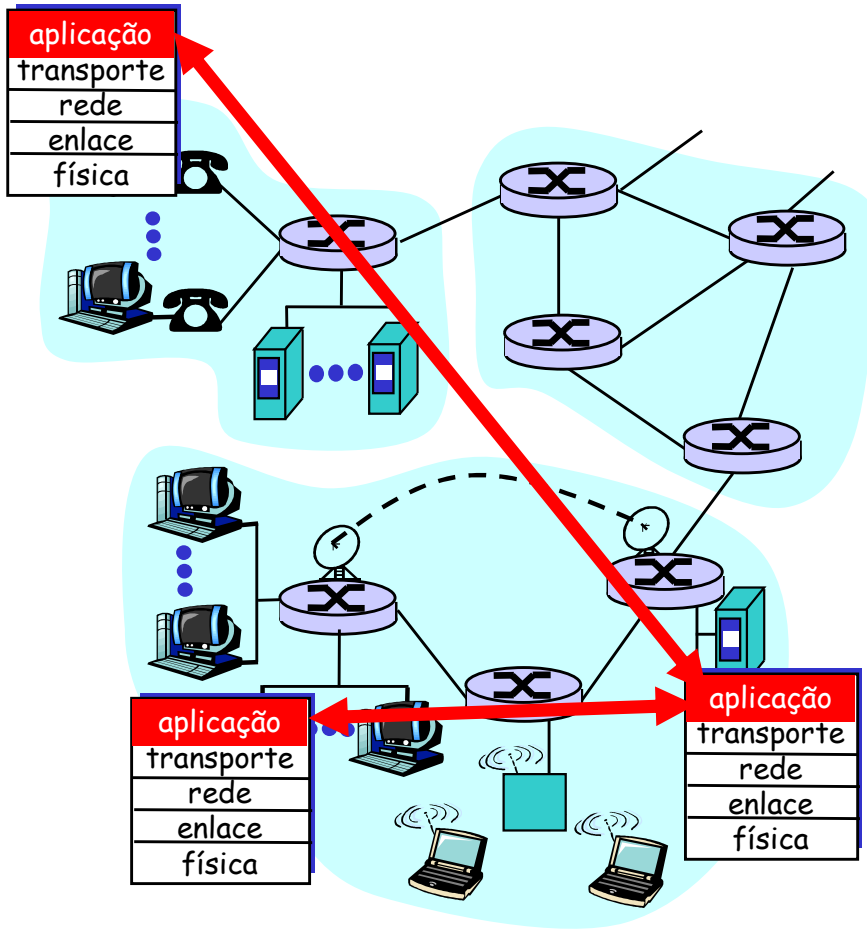
Criando uma aplicação de rede

Programas que

- Executam em (diferentes) sistemas finais
- Comunicam-se através da rede
- p.ex., servidor Web se comunica com o navegador

Programas não relacionados ao núcleo da rede

- Dispositivos do núcleo da rede não executam aplicações dos usuários
- Aplicações nos sistemas finais permite rápido desenvolvimento e disseminação



Arquiteturas das aplicações de rede

- Estruturas possíveis das aplicações:
  - Cliente-servidor
  - Peer-to-peer (P2P)

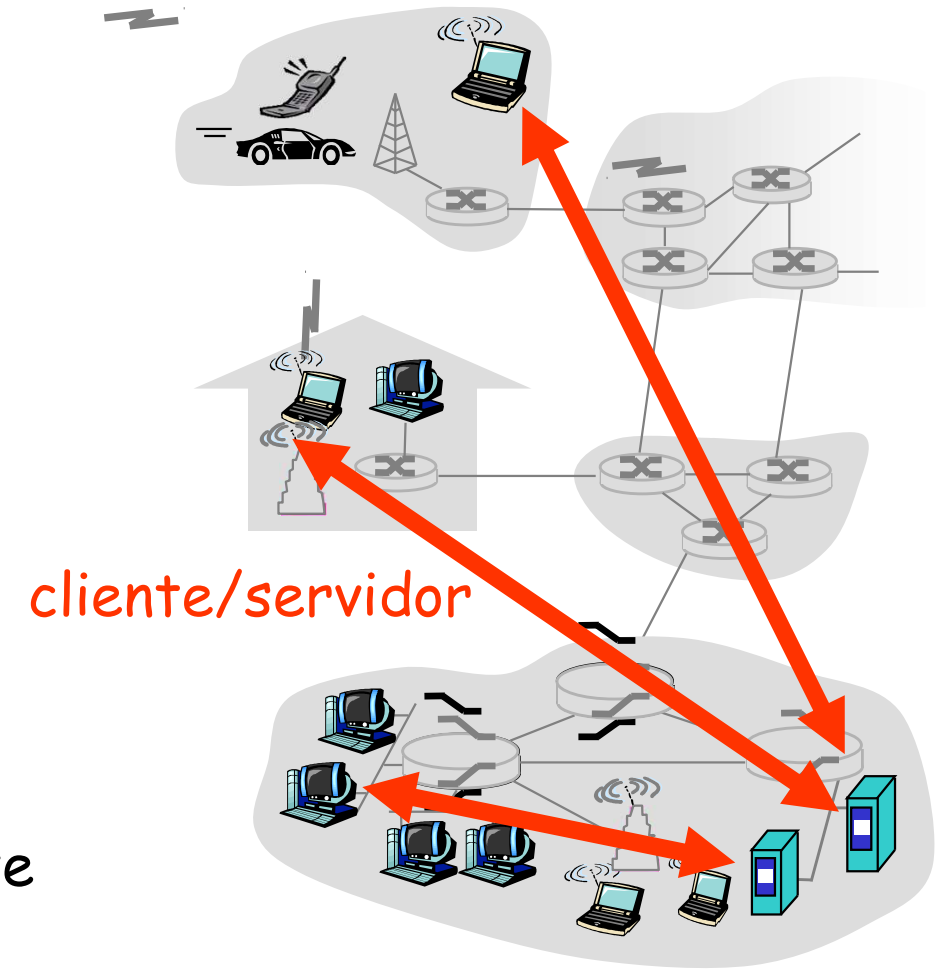
Arquitetura cliente-servidor

Servidor:

- Sempre ligado
- Endereço IP permanente
- Escalabilidade com *data centers*

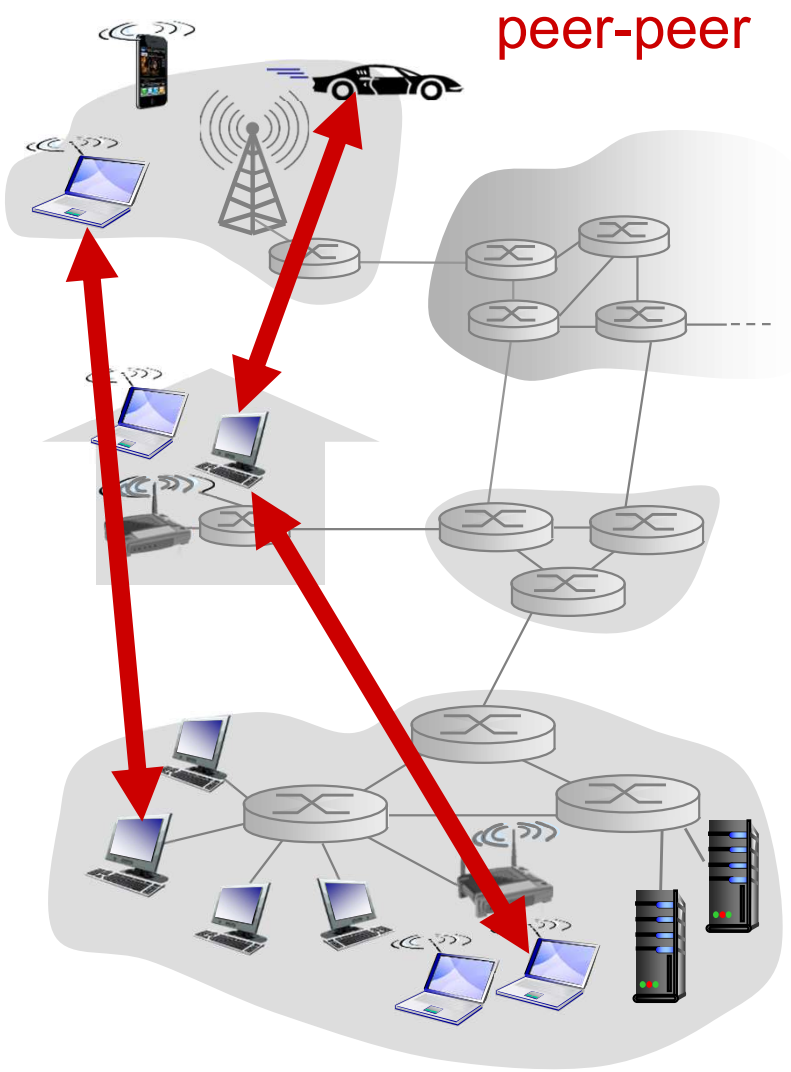
Clientes:

- Comunicam-se com o servidor
- Podem estar conectados intermitentemente
- Podem ter endereços IP dinâmicos
- Não se comunicam diretamente com outros clientes



Arquitetura P2P

- Não há servidor sempre ligado
- Sistemas finais arbitrários se comunicam diretamente
- Pares solicitam serviços de outros pares e em troca proveem serviços para outros parceiros:
  - Autoescalabilidade – novos pares trazem nova capacidade de serviço assim como novas demandas por serviços.
- Pares estão conectados intermitentemente e mudam endereços IP
  - Gerenciamento complexo



Comunicação entre Processos

Processo: programa que executa num sistema final

- processos no mesmo sistema final se comunicam usando comunicação interprocessos (definida pelo sistema operacional)
- processos em sistemas finais distintos se comunicam trocando mensagens através da rede

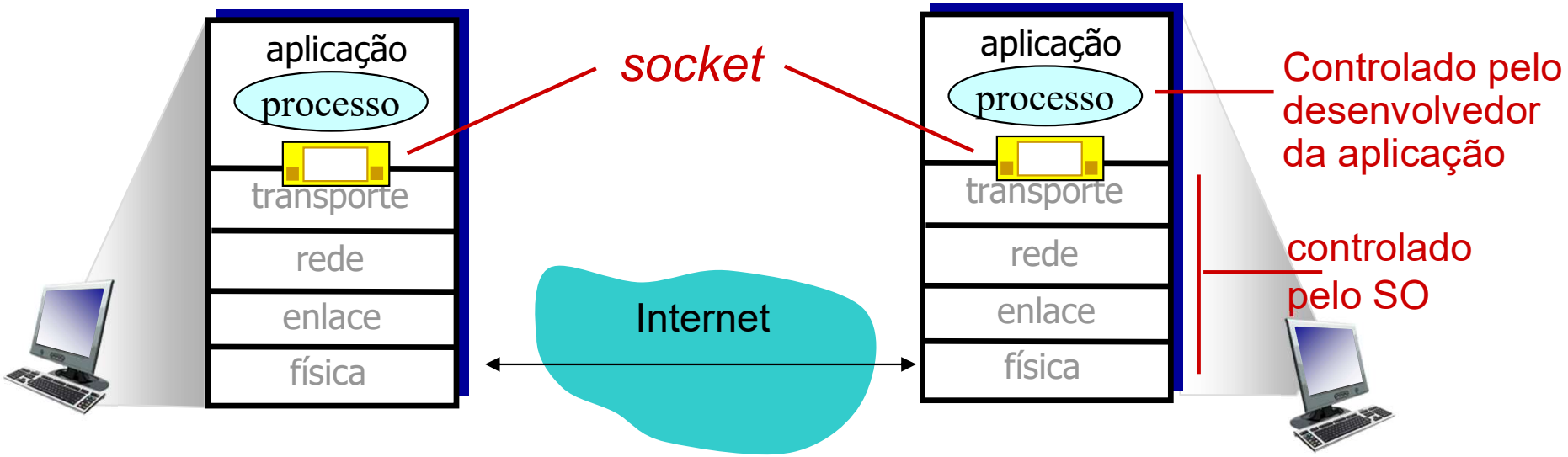
Processo cliente:
processo que inicia a comunicação

Processo servidor:
processo que espera ser contactado

- Nota: aplicações com arquiteturas P2P possuem processos clientes e processos servidores

Sockets

- Os processos enviam/ recebem mensagens para/dos seus *sockets*
- *Um socket é análogo a uma porta*
  - Processo transmissor envia a mensagem através da porta
  - O processo transmissor assume a existência da infra-estrutura de transporte no outro lado da porta que faz com que a mensagem chegue ao *socket* do processo receptor



Endereçamento de processos

- Para que um processo receba mensagens, ele deve possuir um identificador
- Cada hospedeiro possui um endereço IP único de 32 bits
- P: o endereço IP do hospedeiro no qual o processo está sendo executado é suficiente para identificar o processo?
- Resposta: Não, muitos processos podem estar executando no mesmo hospedeiro

- O identificador inclui tanto o endereço IP quanto os números das portas associadas com o processo no hospedeiro .
- Exemplo de números de portas:
  - Servidor HTTP: 80
  - Servidor de Correio: 25
- Para enviar uma msg HTTP para o servidor Web gaia.cs.umass.edu
  - Endereço IP: 128.119.245.12
  - Número da porta: 80
- Mais sobre isto posteriormente.

Os protocolos da camada de aplicação definem

- Tipos de mensagens trocadas:
  - ex. mensagens de requisição e resposta
- Sintaxe das mensagens:
  - campos presentes nas mensagens e como são identificados
- Semântica das msgs:
  - significado da informação nos campos
- Regras para quando os processos enviam e respondem às mensagens

Protocolos abertos:

- definidos em RFCs
- Permitem a interoperação
- ex, HTTP e SMTP

Protocolos proprietários:

- Ex., Skype

De que serviços uma aplicação necessita?

Transferência confiável de dados (sensibilidade a perdas)

- algumas apls (p.ex., transf. de arquivos, transações web) requerem uma transferência 100% confiável
- outras (p.ex. áudio) podem tolerar algumas perdas

Temporização (sensibilidade a atrasos)

- algumas apls (p.ex., telefonia Internet, jogos interativos) requerem baixo retardo para serem "viáveis"

Vazão (largura de banda)

- algumas apls (p.ex., multimídia) requerem quantia mínima de vazão para serem "viáveis"
- outras apls ("apls elásticas") conseguem usar qq quantia de banda disponível

Segurança

- Criptografia, integridade dos dados, ...

Requisitos de aplicações de rede selecionadas

Aplicação	Perdas	Largura de Banda	Sensibilidade ao atraso
transferência de arqs	sem perdas	elástica	não
correio	sem perdas	elástica	não
documentos Web	sem perdas	elástica	não
áudio/vídeo em tempo real	tolerante	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sim, 100's mseg
áudio/vídeo gravado	tolerante	Igual acima	sim, alguns segs
jogos interativos	tolerante	Alguns kbps-10Mbps	sim, 100's mseg
mensagem instantânea	sem perdas	elástica	sim e não

Serviços providos pelos protocolos de transporte da Internet

Serviço TCP:

- *transporte confiável* entre processos remetente e receptor
- *controle de fluxo:* remetente não vai "afogar" receptor
- *controle de congestionamento:* estrangular remetente quando a rede estiver carregada
- *não provê:* garantias temporais ou de banda mínima
- *orientado a conexão:* apresentação requerida entre cliente e servidor

Serviço UDP:

- transferência de dados não confiável entre processos remetente e receptor
- não provê: estabelecimento da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima

P: Qual é o interesse em ter um UDP?

Apls Internet: seus protocolos e seus protocolos de transporte

Aplicação	Protocolo da camada de apl	Protocolo de transporte usado
correio eletrônico	SMTP [RFC 2821]	TCP
acesso terminal remoto	telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferência de arquivos	FTP [RFC 959]	TCP
streaming multimídia	HTTP (ex. Youtube) RTP [RFC 1889]	TCP ou UDP
telefonia Internet	SIP, RTP, proprietário (ex., Skype)	TCP ou UDP

Tornando o TCP seguro

TCP & UDP

- Sem criptografia
- Senhas em texto aberto enviadas aos sockets atravessam a Internet em texto aberto

SSL

- Provê conexão TCP criptografada
- Integridade dos dados
- Autenticação do ponto terminal

SSL está na camada de aplicação

- Aplicações usam bibliotecas SSL, que "falam" com o TCP

API do socket SSL

- Senhas em texto aberto enviadas ao socket atravessam a rede criptografadas
- Vide Capítulo 7

Capítulo 2: Roteiro

- 2.1 Princípios de aplicações de rede
- 2.2 A Web e o HTTP
- 2.3 Transferência de arquivo: FTP
- 2.4 Correio Eletrônico na Internet
- 2.5 DNS: o serviço de diretório da Internet
- 2.6 Aplicações P2P
- 2.7 Programação e desenvolvimento de aplicações com TCP
- 2.8 Programação de *sockets* com UDP

A Web e o HTTP

Primeiro uma revisão...

- Páginas Web consistem de objetos
- um objeto pode ser um arquivo HTML, uma imagem JPEG, um applet Java, um arquivo de áudio,...
- Páginas Web consistem de um arquivo base HTML que inclui vários objetos referenciados
- Cada objeto é endereçável por uma URL
- Exemplo de URL:

`www.someschool.edu/someDept/pic.gif`

nome do hospedeiro | nome do caminho

Protocolo HTTP

HTTP: *hypertext transfer protocol*

- protocolo da camada de aplicação da Web
- modelo cliente/servidor
  - *cliente:* *browser* que pede, recebe (usando o protocolo HTTP) e "visualiza" objetos Web
  - *servidor:* servidor Web envia (usando o protocolo HTTP) objetos em resposta a pedidos

Mais sobre o protocolo HTTP

Usa serviço de transporte TCP:

- cliente inicia conexão TCP (cria *socket*) ao servidor, porta 80
- servidor aceita conexão TCP do cliente
- mensagens HTTP (mensagens do protocolo da camada de apl) trocadas entre *browser* (cliente HTTP) e servidor Web (servidor HTTP)
- encerra conexão TCP

HTTP é "sem estado"

- servidor não mantém informação sobre pedidos anteriores do cliente

Nota

Protocolos que mantêm "estado" são complexos!

- história passada (estado) tem que ser guardada
- Caso caia servidor/cliente, suas visões do "estado" podem ser inconsistentes, devem ser reconciliadas

Conexões HTTP

HTTP não persistente

- No máximo um objeto é enviado numa conexão TCP
  - A conexão é então encerrada
- Baixar múltiplos objetos requer o uso de múltiplas conexões

HTTP persistente

- Múltiplos objetos podem ser enviados sobre uma única conexão TCP entre cliente e servidor

Exemplo de HTTP não persistente

Supomos que usuário digita a URL
www.algumaUniv.br/algumDepartmento/inicial.index

(contém texto, referências a 10 imagens jpeg)

1a. Cliente http inicia conexão TCP a servidor http (processo) a www.algumaUniv.br. Porta 80 é padrão para servidor http.

1b. servidor http no hospedeiro www.algumaUniv.br espera por conexão TCP na porta 80. "aceita" conexão, avisando ao cliente

2. cliente http envia *mensagem de pedido* de http (contendo URL) através do socket da conexão TCP. A mensagem indica que o cliente deseja receber o objeto algumDepartamento/inicial.index

3. servidor http recebe mensagem de pedido, formula *mensagem de resposta* contendo objeto solicitado e envia a mensagem via socket

tempo

Exemplo de HTTP não persistente (cont.)

4. servidor http encerra conexão TCP .

5. cliente http recebe mensagem de resposta contendo arquivo html, visualiza html. Analisando arquivo html, encontra 10 objetos jpeg referenciados

6. Passos 1 a 5 repetidos para cada um dos 10 objetos jpeg

tempo

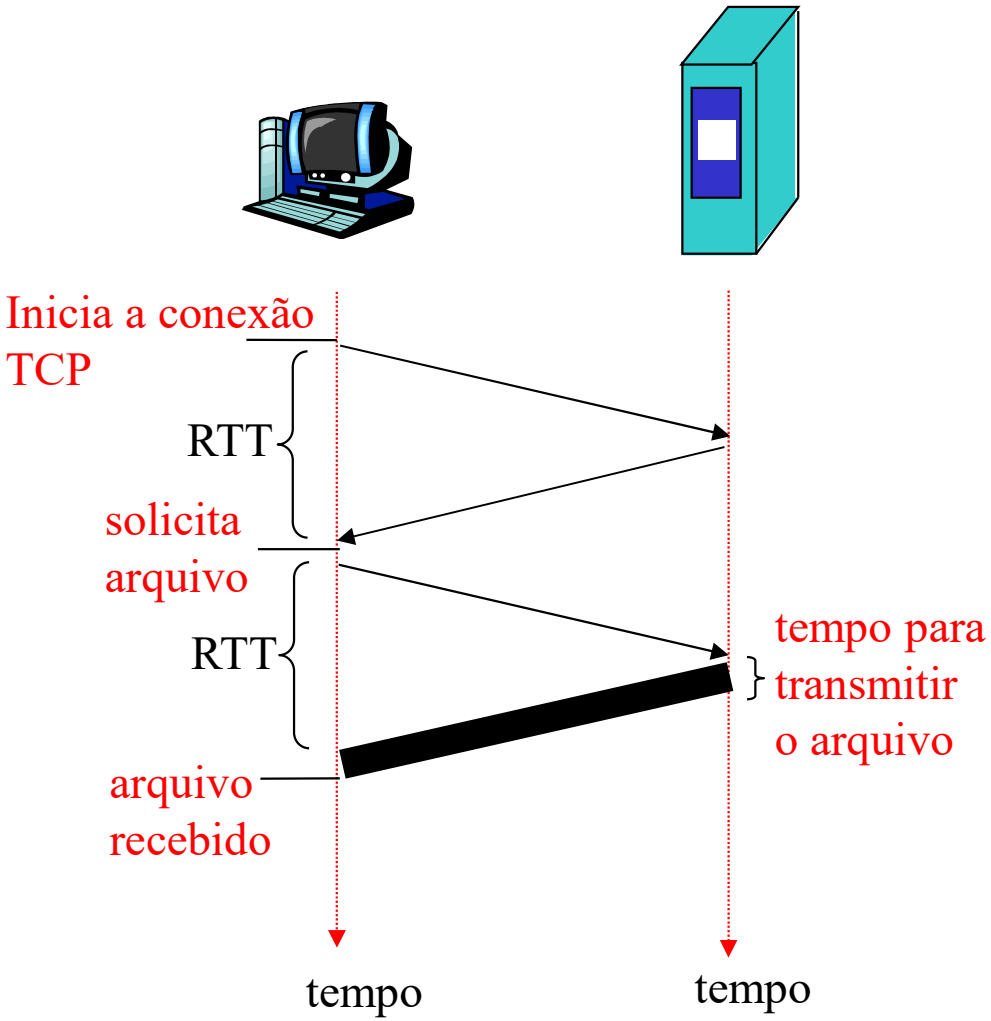
Modelagem do tempo de resposta

Definição de RTT (*Round Trip Time*): intervalo de tempo entre a ida e a volta de um pequeno pacote entre um cliente e um servidor

Tempo de resposta:

- um RTT para iniciar a conexão TCP
- um RTT para o pedido HTTP e o retorno dos primeiros bytes da resposta HTTP
- tempo de transmissão do arquivo

total = 2RTT+tempo de transmissão do arquivo



HTTP persistente

Problemas com o HTTP não persistente:

- requer 2 RTTs para cada objeto
- SO aloca recursos do hospedeiro (*overhead*) para cada conexão TCP
- os *browser* freqüentemente abrem conexões TCP paralelas para recuperar os objetos referenciados

HTTP persistente

- o servidor deixa a conexão aberta após enviar a resposta
- mensagens HTTP seguintes entre o mesmo cliente/servidor são enviadas nesta conexão aberta
- o cliente envia os pedidos logo que encontra um objeto referenciado
- pode ser necessário apenas um RTT para todos os objetos referenciados

Mensagem de requisição HTTP

- Dois tipos de mensagem HTTP: *requisição, resposta*
- mensagem de requisição HTTP:
  - ASCII (formato legível por pessoas)

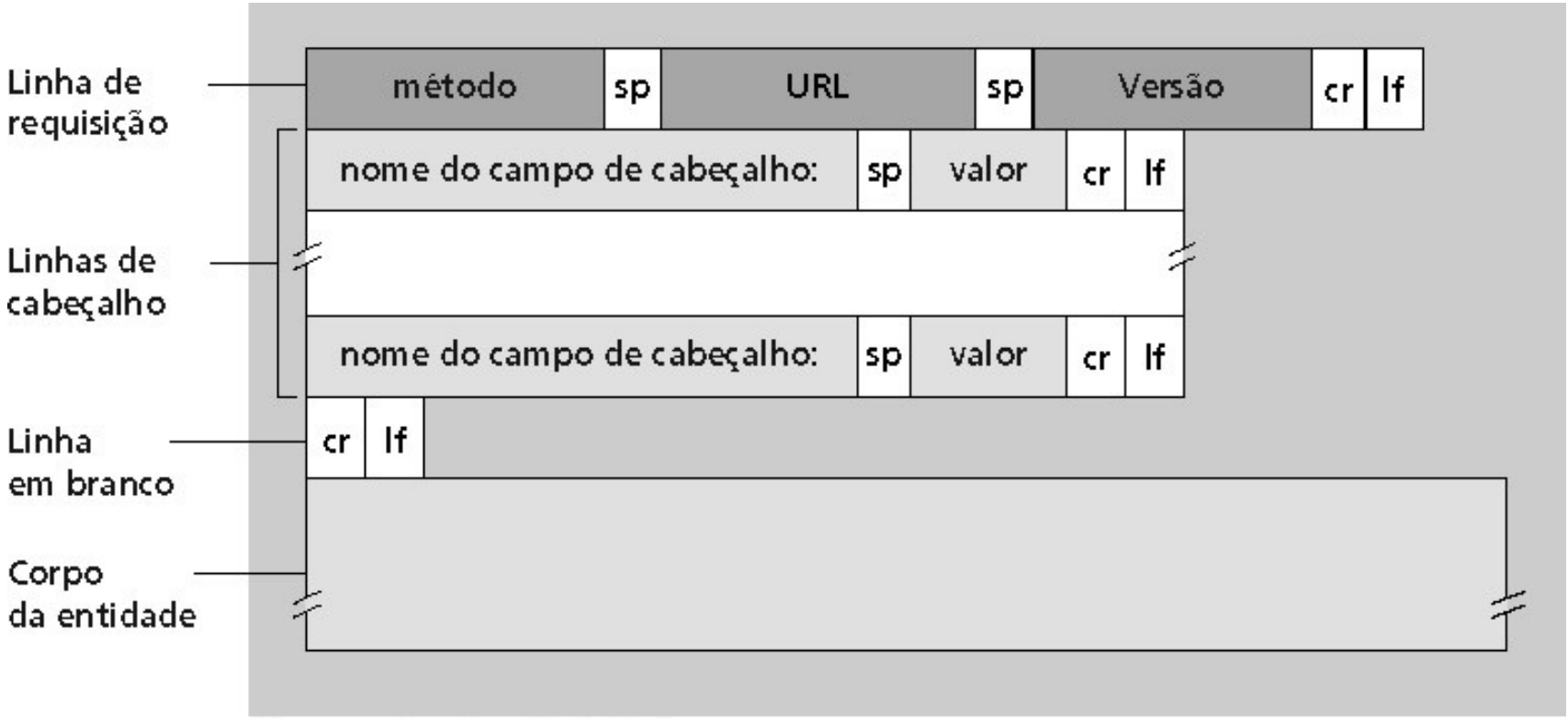
linha da requisição (comandos GET, POST, HEAD)

linhas de cabeçalho

Carriage return, line feed indicam fim de mensagem

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Mensagem de requisição HTTP: formato geral



Obs.: cr = carriage return; lf = line feed

Enviando conteúdo de formulário

Método POST :

- Páginas Web freqüentemente contêm formulário de entrada
- Conteúdo é enviado para o servidor no corpo da mensagem

Método URL:

- Usa o método GET
- Conteúdo é enviado para o servidor no campo URL:

```
www.somesite.com/animalsearch?key=monkeys&bananas
```

Tipos de métodos

HTTP/1.0

- GET
- POST
- HEAD
  - Pede para o servidor não enviar o objeto requerido junto com a resposta

HTTP/1.1

- GET, POST, HEAD
- PUT
  - *Upload* de arquivo contido no corpo da mensagem para o caminho especificado no campo URL
- DELETE
  - Exclui arquivo especificado no campo URL

Mensagem de resposta HTTP

linha de status (protocolo, código de status, frase de status)

linhas de cabeçalho

dados, p.ex., arquivo html solicitado

```
HTTP/1.1 200 OK\r\n
Connection close\r\n
Date: Thu, 07 Jul 2007 12:00:15 GMT\r\n
Server: Apache/1.3.0 (Unix)\r\n
Last-Modified: Sun, 6 May 2007 09:23:24 GMT\r\n
Content-Length: 6821\r\n
Content-Type: text/html\r\n
\r\n
dados dados dados dados ...
```

códigos de status da resposta HTTP

Na primeira linha da mensagem de resposta servidor->cliente. Alguns códigos típicos:

200 OK

- sucesso, objeto pedido segue mais adiante nesta mensagem

301 Moved Permanently

- objeto pedido mudou de lugar, nova localização especificado mais adiante nesta mensagem (Location:)

400 Bad Request

- mensagem de pedido não entendida pelo servidor

404 Not Found

- documento pedido não se encontra neste servidor

505 HTTP Version Not Supported

- versão de http do pedido não usada por este servidor

Experimente você com HTTP (do lado cliente)

1. Use cliente telnet para seu servidor WWW favorito:

```
telnet cis.poly.edu 80
```

Abre conexão TCP para a porta 80 (porta padrão do servidor http) a www.ic.uff.br. Qualquer coisa digitada é enviada para a porta 80 do www.ic.uff.br

2. Digite um pedido GET HTTP:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

Digitando isto (deve teclar ENTER duas vezes), está enviando este pedido GET mínimo (porém completo) ao servidor http

3. Examine a mensagem de resposta enviada pelo servidor HTTP !

(ou use Wireshark para ver as msgs de pedido/resposta HTTP capturadas)

Cookies: manutenção do "estado" da conexão

Muitos dos principais sítios Web usam *cookies*

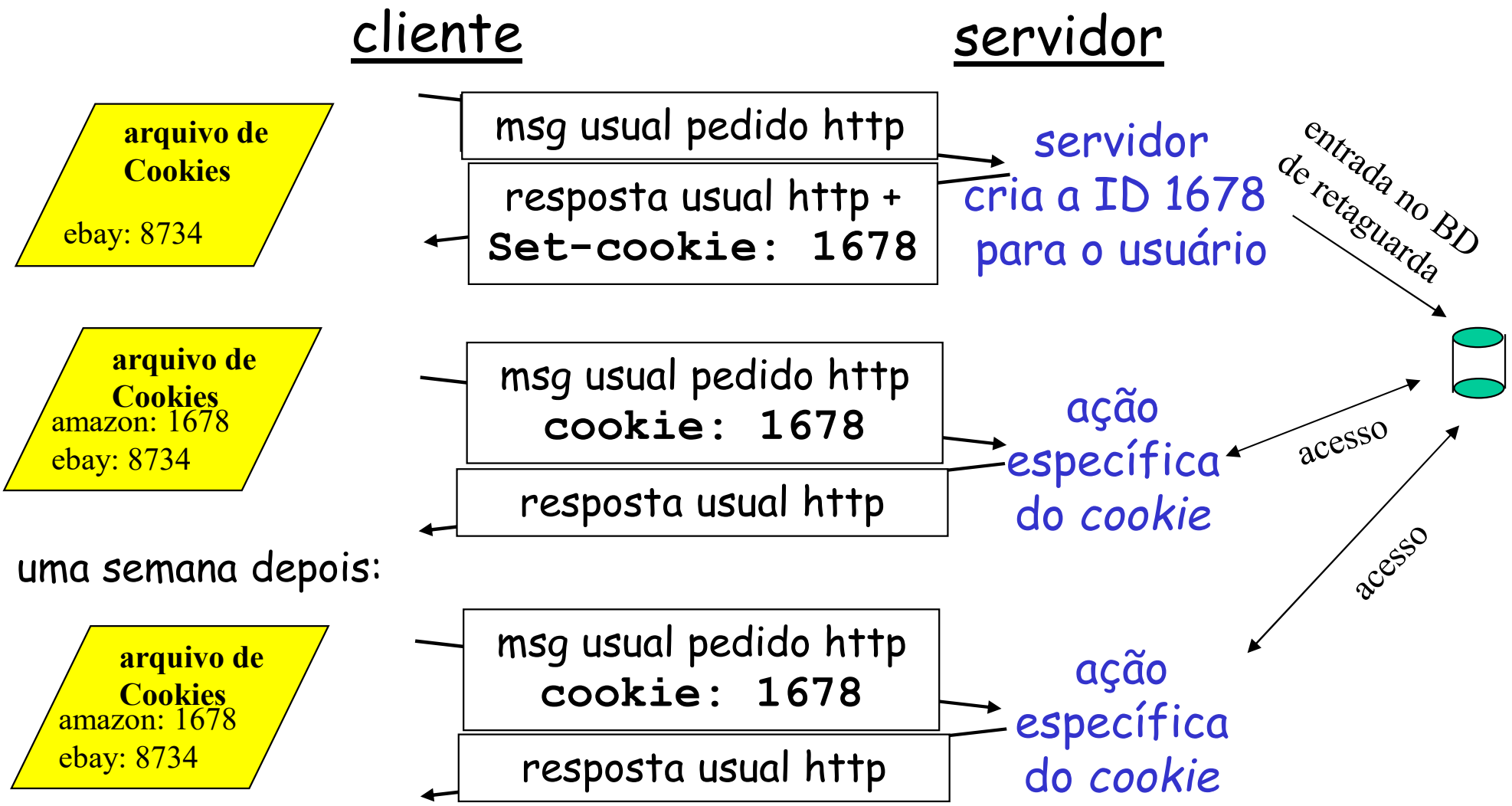
Quatro componentes:

1) linha de cabeçalho do *cookie* na mensagem de resposta HTTP
2) linha de cabeçalho do *cookie* na mensagem de pedido HTTP
3) arquivo do *cookie* mantido no host do usuário e gerenciado pelo browser do usuário
4) BD de retaguarda no sítio Web

Exemplo:

- Suzana acessa a Internet sempre do mesmo PC
- Ela visita um sítio específico de comércio eletrônico pela primeira vez
- Quando os pedidos iniciais HTTP chegam no sítio, o sítio cria
  - uma ID única
  - uma entrada para a ID no BD de retaguarda

Cookies: manutenção do "estado" (cont.)



Cookies (continuação)

O que os *cookies* podem obter:

- autorização
- carrinhos de compra
- recomendações
- estado da sessão do usuário (*Webmail*)

nota

Cookies e privacidade:

- cookies permitem que os sítios aprendam muito sobre você
- você pode fornecer nome e e-mail para os sítios

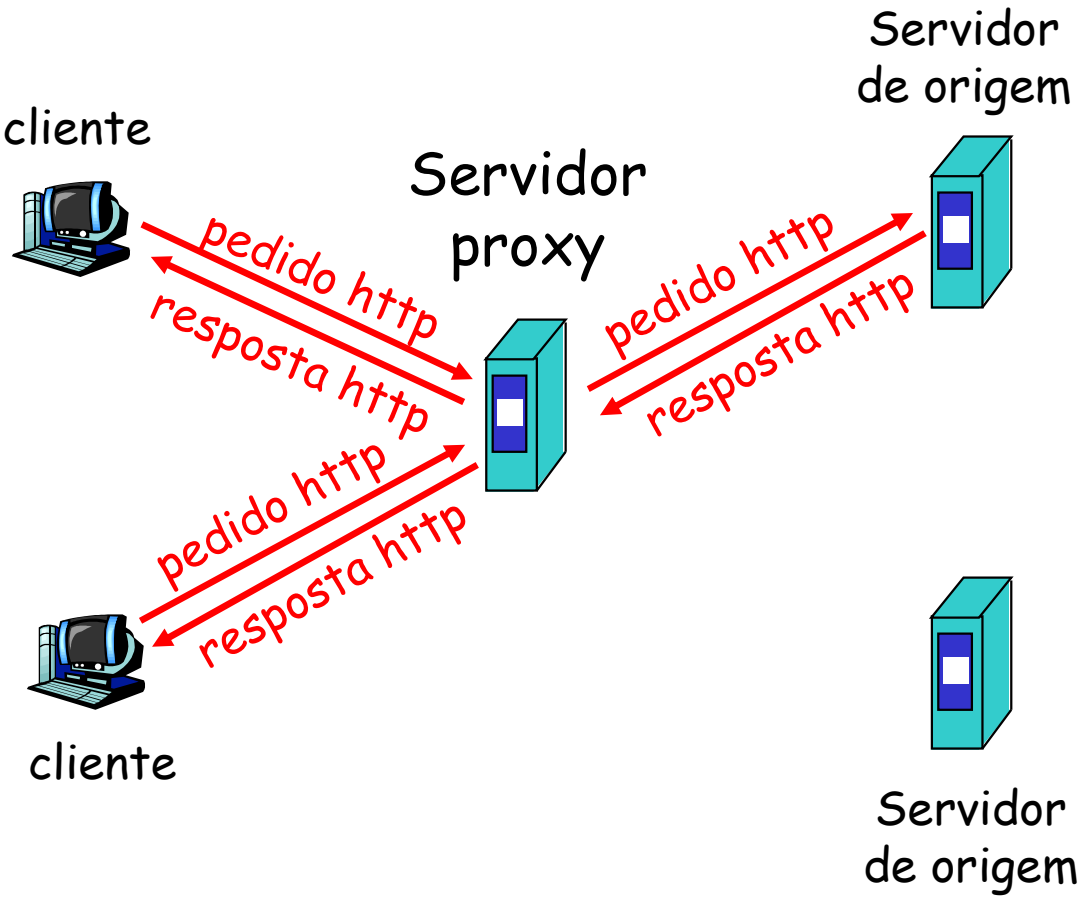
Como manter o "estado":

- Pontos finais do protocolo: mantêm o estado no transmissor/receptor para múltiplas transações
- *Cookies*: mensagens http transportam o estado

Cache Web (servidor proxy)

Meta: atender pedido do cliente sem envolver servidor de origem

- usuário configura browser: acessos Web via proxy
- cliente envia todos pedidos HTTP ao proxy
  - se objeto estiver no cache do proxy, este o devolve imediatamente na resposta HTTP
  - senão, solicita objeto do servidor de origem, depois devolve resposta HTTP ao cliente



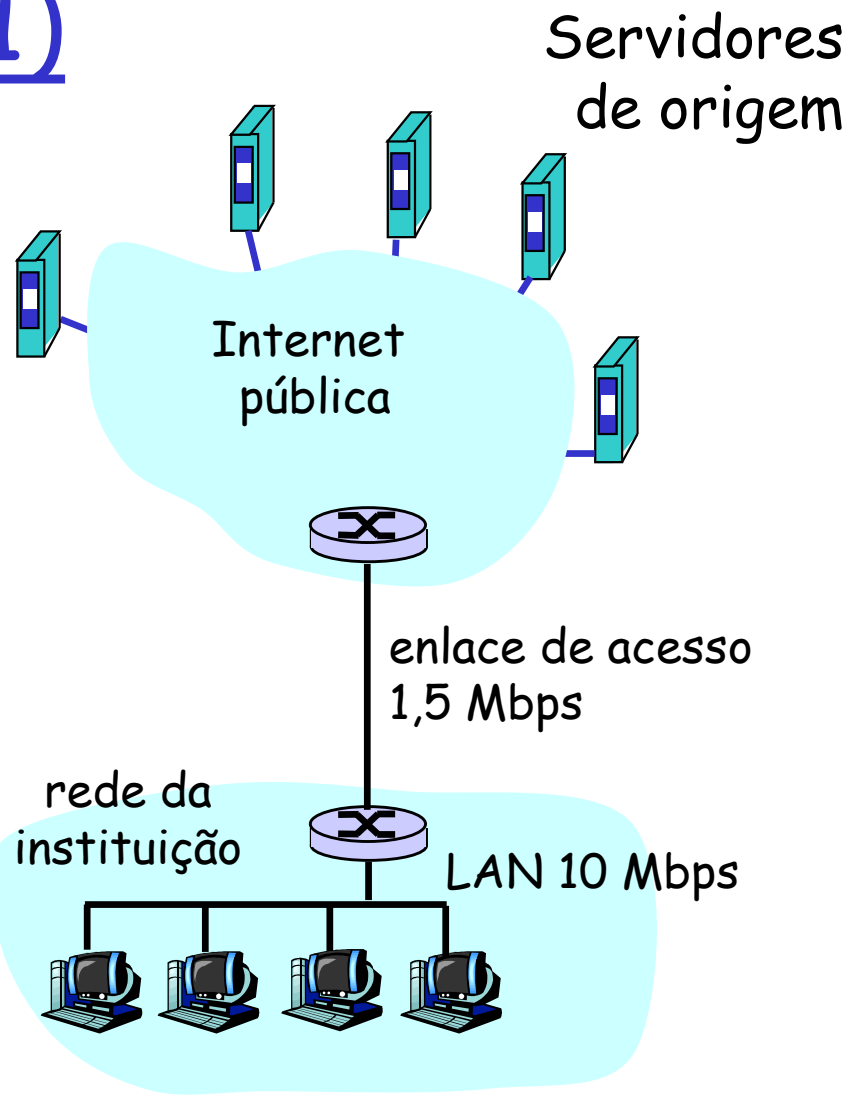
Mais sobre Caches Web

- Cache atua tanto como cliente quanto como servidor
- Tipicamente o cache é instalado por um ISP (universidade, empresa, ISP residencial)

Para que fazer cache Web?

- Redução do tempo de resposta para os pedidos do cliente
- Redução do tráfego no canal de acesso de uma instituição
- A Internet cheia de caches permitem que provedores de conteúdo "pobres" efetivamente forneçam conteúdo (mas o compartilhamento de arquivos P2P também!)

Exemplo de cache (1)



Hipóteses

- Tamanho médio de um objeto = 100.000 bits
- Taxa média de solicitações dos *browsers* de uma instituição para os servidores originais = 15/seg
- Atraso do roteador institucional para qualquer servidor origem e de volta ao roteador = 2seg

Conseqüências

- Utilização da LAN = 15%
- Utilização do canal de acesso = 100% *problema!*
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + minutos + microsegundos

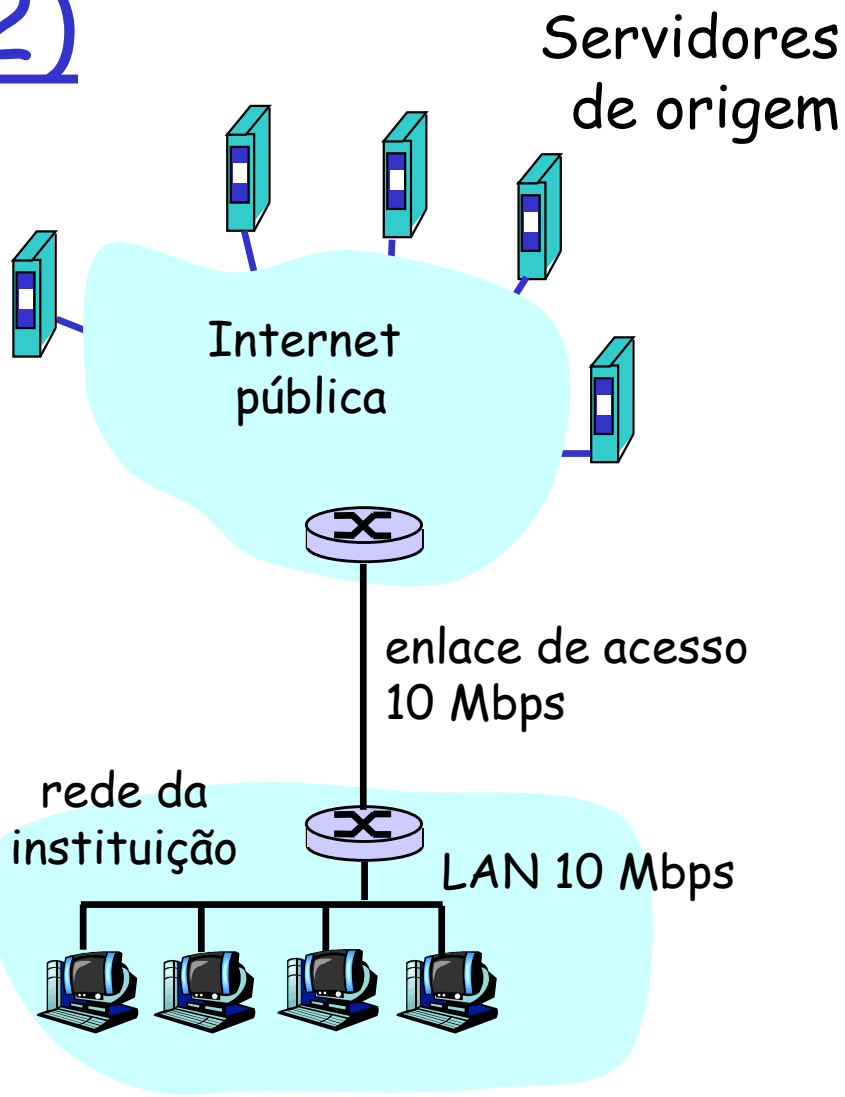
Exemplo de cache (2)

Solução em potencial

- Aumento da largura de banda do canal de acesso para, por exemplo, 10 Mbps

Conseqüências

- Utilização da LAN = 15%
- Utilização do canal de acesso = 15%
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 2 seg + msegs + msegs
- Frequentemente este é uma ampliação cara



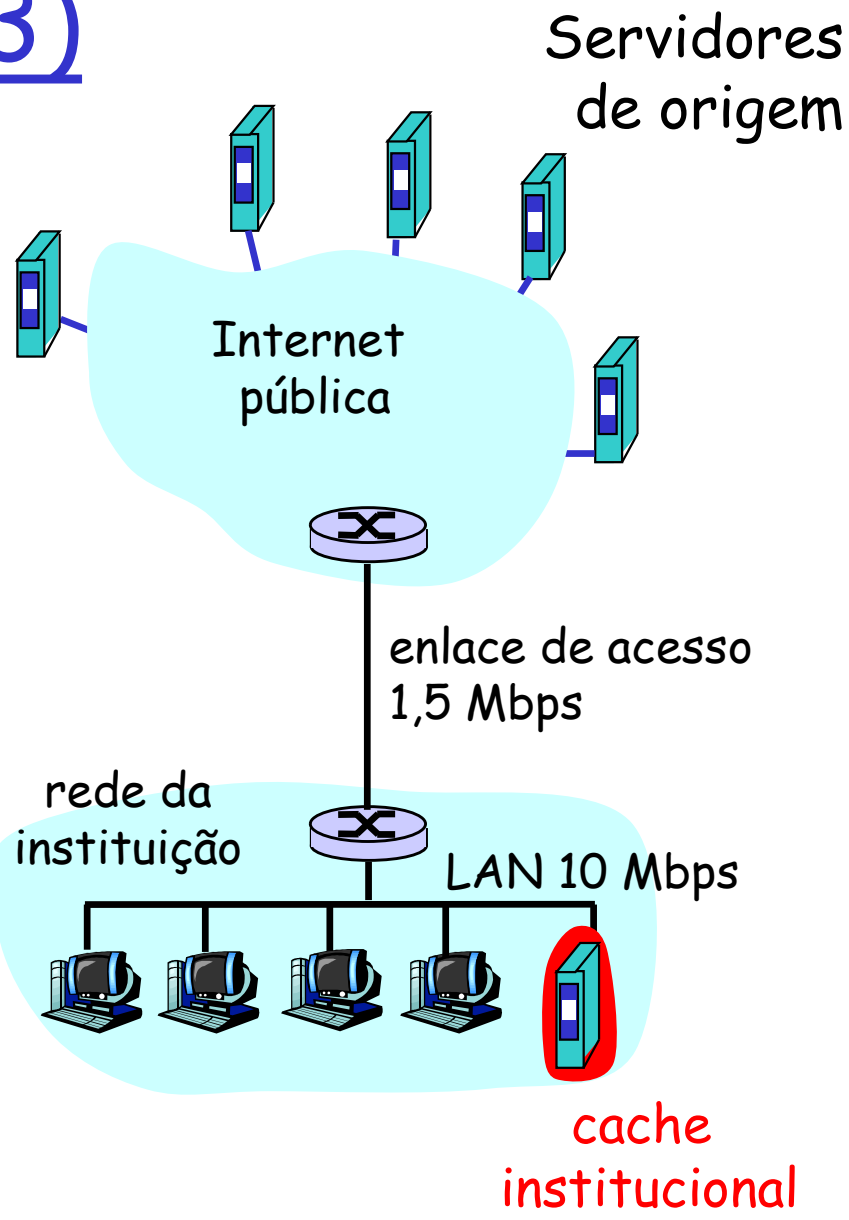
Exemplo de cache (3)

Instale uma cache

- Assuma que a taxa de acerto seja de 0,4

Conseqüências

- 40% dos pedidos serão atendidos quase que imediatamente
- 60% dos pedidos serão servidos pelos servidores de origem
- Utilização do canal de acesso é reduzido para 60%, resultando em atrasos desprezíveis (ex. 10 mseg)
- Atraso total = atraso da Internet + atraso de acesso + atraso na LAN = 0,6*2 seg + 0,6*0,01 segs + msegs < 1,3 segs



GET condicional

- Meta: não enviar objeto se cliente já tem (no cache) versão atual
  - Sem atraso para transmissão do objeto
  - Diminui a utilização do enlace
- cache: especifica data da cópia no cache no pedido HTTP

```
If-modified-since:
  <date>
```

- servidor: resposta não contém objeto se cópia no cache for atual:

```
HTTP/1.0 304 Not
  Modified
```